Logical conditional preference theories

Cristina Cornelio[1], Andrea Loreggia[12], and Vijay Saraswat[2]

[1] Padua University, Padua, Italy
{cornelio,loreggia}@math.unipd.it
[2] IBM T.J. Watson Research Center, Yorktown Heights, New York, U.S.
vijay@saraswat.org

Abstract. CP-nets represent the dominant existing framework for expressing qualitative conditional preferences between alternatives, and are used in a variety of areas including constraint solving. Over the last fifteen years, a significant literature has developed exploring semantics, algorithms, implementation and use of CP-nets.
This paper introduces a comprehensive new framework for conditional preferences: *logical conditional preference theories* (LCP theories). To express preferences, the user specifies arbitrary (constraint) Datalog programs over a binary ordering relation on outcomes. We show how LCP theories unify and generalize existing conditional preference proposals, and leverage the rich semantic, algorithmic and implementation frameworks of Datalog.

Keywords: CP-nets, preferences, datalog, constraint logic programming

1 Introduction

Qualitative conditional preferences on combinatorial domains are an important area of study. The main framework is *CP-nets* [BBD+04b]. Given a finite set of features, each with a finite domain of values, the user specifies her preference order over the domain of each feature using rules such as $a\,b : c \succ \bar{c}$ which specify that if the attribute A has value a and attribute B has value b then c is to be preferred to $\bar{c}$ for attribute C. *Outcomes* (that is, assignment of values to all the features) are ordered according to the so-called *ceteris paribus* interpretation: an outcome s is preferred to another outcome t if they differ on the value of just one feature, and the value in s is preferred to the one in t (given the rest of s, t). Thus in CP-nets, preferences are always of the form "I prefer fish to meat, *all else being equal*". The key computational tasks are checking for consistency (preference ordering is acyclic), as well as optimizing and comparing outcomes.

CP-nets represent one end of the expressiveness/tractability spectrum. While useful in practice, CP-nets are limited in expressiveness: a rule may specify a preference for exactly one value over another (in the same feature domain). *General CP-nets* [GLTW08] define CP-nets that can be incomplete or locally inconsistent. *CP-theories* [Wil04] are a specialized formalism, with its own *ad hoc* syntax (rules are of the form $u : x \succ x'\,[W]$) and semantics, in which preferences may be conditioned on *indifference* to certain features (those in the set W above). For example, one may say "I prefer fish to meat, no matter what the dessert is". *Comparative preference theories* [Wil09] permit preferences to be defined on a set of features simultaneously. Along another direction, [BBD$^+$04a] and [DPR$^+$06] introduce the idea of adding (hard and soft) constraints to CP-nets. Roughly speaking, the added constraints need to be respected by valid outcomes.

This paper develops the fundamental idea that conditional preferences can be directly expressed in standard first order logic, as constrained Datalog theories [CGT89,KKR95,Tom98] involving a binary preference relation `d(_,_)` on pairs of outcomes. For instance, in the context of preferences over entrees and desserts, the rule "I prefer fish to meat, all else being equal" may be written as:[3]

```
d(o(fish,X),o(meat,X)).
```

and the rule "I prefer fish to meat, no matter what the dessert is" is written as:

```
d(o(fish,X),o(meat,Y)).
```

More generally, a *Logical Conditional Preference* (LCP) rule is of the form:

```
d(o(X1,...Xn),o(Y1,...,Yn)) :- c, g1,...,gn.
```

where `c` is a constraint (possibly involving equalities), and `g1`, ..., `gn` are possibly recursively defined predicates involving `d/2`. A predicate `dom/2` is defined to be the transitive closure of `d/2` and represents the dominance relation on outcomes.

```
dom(X,Z):- d(X,Z), outcome(X), outcome(Z).
dom(X,Z):- d(X,Y), dom(Y,Z).
```

The theory can be checked for consistency by simply ensuring that given the clause

```
inconsistent :- dom(X,X).
```

the goal `inconsistent` cannot be established. Hard constraints `C` are specified by adding them to the body of the clause defining legal outcomes:

```
outcome(o(X1,...,Xn)):- C, d1(X1), ..., dn(Xn).
```

The fundamental advantage of introducing conditional preference theories as Datalog programs is that Datalog's rich semantic, algorithmic and implementation framework is now available in service of conditional preferences. The semantics of LCP theories is that of (constrained) first-order logic theories. The framework is rich enough to express CP-nets and each of its extensions discussed above, including algorithms for consistency, dominance and optimality

[3] The examples in this paper are written in Prolog and run in the XSB Prolog system, using tabling. Note that the clause is strictly speaking not a Datalog clause because it uses a function symbol `o/n`. However this function symbol is used just for convenience, and can be eliminated at the cost of increasing the arity of predicates such as `d` and `dom`.

(Section 3). Using outcomes rather than assertions of preference over individual features permits the formalization of the semantics (e.g. *ceteris paribus* or indifference) internally, as just a certain pattern of quantification over variables.

Constraints fit in naturally and do not have to be introduced after the fact in an *ad hoc* fashion as in [DPR$^+$06,BBD$^+$04a,PRVW04]. For example, [PRVW04] provides a dominance algorithm using the notion of a *consistent flipping worsening sequence*: they allow worsening flips only between consistent outcomes. In our formulation the constraints are already built into basic definitions (constraints are additional goals in the body of preference clauses, and in the body of the clause defining outcomes) and no changes are necessary.

Additionally, recursive LCP-rules (rules with goals `gi` in the body) offer a powerful new form of *dependent* conditional preference statements (Section 3.1), particularly useful in multi-agent contexts [RVW04,MMP$^+$13]. They support rules such as "If Alice prefers to drive to Oxford today, Bob will prefer to fly to Manchester tomorrow".

The rich complexity theory developed for Datalog [Var82,FV99,GP03,DEGV01] applies *inter alia* to conditional preference theories – in particular we discuss the notion of *data-complexity* in Section 2.3. General results about (linear) Datalog programs lead to complexity bounds for consistency, dominance and optimization extending current known bounds, and in some cases, providing new, simpler proofs for existing complexity bounds (Section 4). Further, tabled Prolog systems such as XSB Prolog[SW10,SW12a] implement constrained Datalog with sophisticated features such as partial order answer subsumption that are directly usable in an implementation of LCP.

One of the reasons that CP-nets are popular in practice is that useful special cases have been identified (acyclic nets, tree-structured nets) which can be implemented efficiently. In Section 4 we show how some of these special cases can be extended to the richer language we consider. Further, we provide a compiler for LCP theories that can recognize these special cases and generate custom code for consistency, dominance and optimization (Section 5). We present some scalability numbers.

In summary, we believe our formalization of extensions of CP-nets permits an integrated treatment of preferences in constraint (logic) programming, leading to more powerful reasoning systems which can deal with both preferences and hard constraints.

Rest of the paper. Section 2 introduces the models of GCP-nets, CP-nets, CP-theories and comparative preference languages, and Datalog and tabled logic programming, and provides basic computational results. Section 3 specifies the LCP formulation, and establishes that LCP-theories conservatively extend GCP-nets, CP-nets, CP-theories and comparative preference languages. Section 4 studies the computational complexity of outcome optimization, consistency checking and dominance queries. We also provide specific algorithms for the special cases (e.g. acyclic structure and tree structure dependency graphs). Section 5 describes the implementation of the LCP compiler. We conclude in Section 6.

2 Background

Below we assume given a set of N *features* (or variables), $\texttt{Var} = \{X_1, \ldots, X_N\}$. We assume for simplicity that the values in each feature X are either $x, \bar{x}$ (handling multiple values is easy).

2.1 GCP-nets and CP-nets

GCP-nets ([GLTW08] and [DRVW03]) allow a general form of conditional and qualitative preferences to be modeled compactly.

Definition 1. *A* ***Generalized CP-net (GCP-net)*** *C over $\texttt{Var}$ is a set of conditional preference rules. A* **conditional preference rule** *is an expression $p : l > \bar{l}$, where l is a literal of some atom $X \in \texttt{Var}$ and p is a propositional formula over $\texttt{Var}$ that does not involve variable X. A GCP-net corresponds to a directed graph (dependency graph) where each node is associated with a feature and the edges are pairs (Y, X) where Y appears in p in some rule $p : x > \bar{x}$ or $p : \bar{x} > x$. Each node X is associated with a* CP-table *which expresses the user preference over the values of X. Each row of the CP-table corresponds to a conditional preference rule.*

The CP-tables of a GCP-net can be *incomplete* (i.e. for some values of some variables' parents, the preferred value of X may not be specified) and/or *locally inconsistent* (i.e. for some values of some variables' parents, the table may both contain the information $x > \bar{x}$ and $\bar{x} > x$). CP-nets [BBD+04b] are a special case of GCP-net in which the preferences are locally consistent and locally complete.

An *outcome* in a CP-net is a complete assignment to all features. For example, given $\texttt{Var} = \{X_1, X_2\}$ and binary domains $D_1 = D_2 = \{T, F\}$, all the possible outcomes are TT, TF, FT and FF.

A *worsening flip* is a change in the value of a feature to a value which is less preferred according to the cp-statement for that feature. This concept defines an order over the set of outcomes such that one outcome o is *preferred* to another outcome o' ($o \succ o'$) if and only if there is a chain of worsening flips from o to o'. The notion of *worsening flip* induces a preorder over the set of outcomes. This preorder allows maximal elements that correspond to the so-called *optimal outcomes*, which are outcomes that have no other outcome better than them.

Given any GCP-net and CP-net the problems of consistency checking and finding optimal outcomes are PSPACE-complete [GLTW08]. Moreover, there could be several different maximal elements. When the dependency graph has no cycle the CP-net is called *acyclic*. The optimal outcomes for such nets are unique and can be found in polynomial time in N. The procedure used to this purpose is usually called a *sweep forward* and takes N steps [BBD+04b].

The problem of dominance testing (i.e. determining if one outcome is preferred to another) is PSPACE-complete for both GCP-nets and CP-nets. It is polynomial if the CP-nets are tree structured or poly-tree structured [DB02,GLTW08].

2.2 CP-theories and comparative preference languages

CP-theories are introduced in [Wil04] as a logic of conditional preference which generalizes CP-nets.

Definition 2. *Given a set of variables* $\mathtt{Var} = \{X_1, \cdots, X_N\}$ *with domains* D_i, $i = 1, \ldots, n$, *the language* $L_{\mathtt{Var}}$ *is defined by all the statements of the form:* $u : x_i \succ x'_i[W]$ *where* u *is an assignment of a set of variables* $U \subseteq \mathtt{Var} \setminus \{X_i\}$, $x_i \neq x'_i \in D_i$ *and* W *is a set of variables such that* $W \subseteq (\mathtt{Var} \setminus U \setminus \{X_i\})$.

Definition 3. *Given a language* $L_{\mathtt{Var}}$ *as defined above, a* conditional preference theory (CP-theory) Γ *on* $\mathtt{Var}$ *is a subset of* $L_{\mathtt{Var}}$. Γ *generates a set of preferences that corresponds to the set* $\Gamma^* = \bigcup_{\varphi \in \Gamma} \varphi^*$ *where given* $\varphi = u : x_i \succ x'_i[W]$, φ^* *is defined as* $\varphi^* = \{(tuxw, tux'w') : t \in \mathtt{Var} \setminus (\{X_i \cup U \cup W\}),\ w, w' \in W\}$.

A CP-net is a particular case of a CP-theory where $W = \emptyset$ for all $\varphi \in \Gamma$.

Two graphs are associated to a CP-theory: $H(\Gamma) = \{(X_j, X_i) | \exists \varphi \in \Gamma \text{ s.t. } \varphi = u : x_i \succ x'_i[W] \text{ and } X_j \in U\}$ and $G(\Gamma) = H \cup \{(X_i, X_j) | \exists \varphi \in \Gamma \text{ s.t. } \varphi = u : x_i \succ x'_i[W] \text{ and } X_j \in W\}$.

The semantics of CP-theories depends on the notion of a *worsening swap*, which is a change in the assignment of a set of variables to an assignment which is less preferred by a rule $\varphi \in \Gamma$. We say that one outcome o is better than another outcome o' ($o \succ o'$) if and only if there is a chain of worsening swaps (a *worsening swapping sequence*) from o to o'.

Definition 4. *A CP-theory* Γ *is* locally consistent *if and only if for all* $X_i \in \mathtt{Var}$ *and* $u \in Pa(X_i)$ *in the graph* $H(\Gamma)$, $\succ_u^{X_i}$ *is irreflexive.*

Local consistency can be determined in time proportional to $|\Gamma|^2 N$. Given a CP-theory Γ, if the graph $G(\Gamma)$ is acyclic, Γ is consistent if and only if Γ is locally consistent, thus global consistency has the same complexity as local consistency given an acyclic graph $G(\Gamma)$.

Comparative preference theories [Wil09] are an extension of CP-theories.

Definition 5. *The comparative preference language* $\mathcal{CL}_{\mathtt{Var}}$ *is defined by all statements of the form:* $p > q || T$ *where* P, Q *and* T *are subsets of* $\mathtt{Var}$ *and* p *and* q *are assignments respectively of the variables in* P *and in* Q.

Definition 6. *Given a language* $\mathcal{CL}_{\mathtt{Var}}$ *as defined above, a* comparative preference theory Λ *on* $\mathtt{Var}$ *is a subset of* $\mathcal{CL}_{\mathtt{Var}}$. Λ *generates a set of preferences that corresponds to the set* $\Lambda^* = \bigcup_{\varphi \in \Lambda} \varphi^*$ *where if* $\varphi = p > q || T$, φ^* *is defined as a pair* (α, β) *of outcomes such that* α *extends* p *and* β *extends* q *and* α *and* β *agree on* T: $\alpha \restriction_T = \beta \restriction_T$.

2.3 Datalog and tabled logic programming

A Datalog program consists of a collection of definite clauses in a language with no function symbols, hence a finite Herbrand domain. Datalog programs can be implemented using *tabled Logic Programming (TLP)*. Tabling maintains a memo table of subgoals produced in a query evaluation and their answers. If a subgoal is reached again then the information in the table can be reused, without recomputing the subgoal. This method ensures termination and improves the computational complexity for a large class of problems [SW12a] (at the expense of additional space consumption). Answer subsumption extends the functionality of tabling. *Answer variance* adds a new answer to a table only if the new answer is not a variant of any other answer already in the table. *Partial order answer subsumption* adds a new answer to a table only if the new answer is maximal with respect to the answers in the table, given a partial order `po/2`:

```
:-  table predicate(_,_,partialOrder(po/2)).
```

Traditionally, predicates are divided into *extensional* and *intensional* predicates. The extensional predicates define a database, and intensional predicates define (possibly recursively defined) queries over the database. In our context, `d/2` and `outcome/1` will be considered extensional predicates (in Flat LCP) and other predicates such as `dom/2, inconsistent` and user-defined predicates are considered intensional. Given an intensional program P, database D and query q, *data-complexity* [Var82] is the complexity of answering $P, D \vdash q$ as a function of the size of D and q (thus the program is considered fixed). *Combined complexity* is the complexity of answering $P, D \vdash q$ as a function of the size of P, D and q (thus nothing is taken to be fixed). The basic results are that for data-complexity, general Datalog programs are PTIME-complete, and linear programs are NLOGSPACE-complete [GP03]. For combined complexity, general Datalog programs are EXPTIME-complete, and linear programs are PSPACE-complete.

3 Logical Conditional Preference Theories

We assume given a set of N features, and a logical vocabulary $\mathcal{V}$ with unary predicates `d1`, ..., `dN` (corresponding to the domains of the features), constants for every value in the domains `di`, a single function symbol `o/N`, and a single binary predicate `d/2`.

The user specifies preferences between two outcomes `S,T` (expressed as `o/N` terms) by supplying clauses for the atom `d(S,T)`:

```
d(S,T) :- C, g1, ..., gk.
```

where `C` is a constraint (possibly involving equality), and `g1`, ..., `gk` are possibly recursively defined predicates involving `d/2`. The clause is said to be *flat* if `k=0`, else it is *recursive*. The user specifies hard constraints on features by providing clauses for the `outcome/1` predicate, typically of the form

```
outcome(o(X1,...,Xn)) :- C, d1(X1), ..., dn(Xn).
```

where `C` is a constraint and the `di` are domain predicates.

The LCP runtime supplies the following definition for the `dom/2` predicate, expressing (tabled) transitive closure over `d/2`, and for consistency and optimal outcomes:

```
:- table(dom(_,_)).
dom(X,Y):- d(X,Y), outcome(X), outcome(Y).
dom(X,Y):- d(X,Z), dom(Z,Y).
consistent :- \+ dom(X,X).
:- table(optimal(po(dom/2))).
optimal(X):- outcome(X).
```

Note that the clauses above are linear. Below, given an LCP theory (Datalog program) P, by $\mathcal{L}(P)$ we will mean P together with the LCP runtime clauses specified above.

Given these definitions, the problem solver may use `consistent` to determine whether the supplied preference clauses are consistent, `dom(S,T)` to determine whether outcome `S` is preferred to `T`, and `optimal(S)` (where `S` may be a partially instantiated `o/N` structure) to determine an optimal completion of `S`.

Example 1 (Dinner, modified from [BBD$^+$04b]). Two components of a meal are the soup (`fish` or `veg`) and wine (`white` or `red`). I prefer `fish` to `veg`. If I am having `fish`, I prefer `white` wine to `red`. I simply do not want to consider `veg` with `red`. This may be formulated as the LCP theory:

```
soup(fish). soup(veg). wine(white). wine(red).
outcome(o(X,Y)):- soup(X), wine(Y), (X\== veg; Y\==red).
d(o(fish, X),    o(veg,  X)).
d(o(fish, white),o(fish, red)).
```

On this theory, the query `?-consistent.` returns `yes`. The `dom/2` predicates order outcomes as:

```
o(fish,white) > o(fish,red)   o(fish,white) >  o(veg,white)
```

Note because of hard constraints the outcomes are not totally ordered. The query `optimal(X)` returns the single answer `X=o(fish,white)`; the query `optimal(o(fish,X))` returns `X=white`; `optimal(o(X,red))` returns `X=fish`, etc. The behavior of the `optimal/1` queries will be explained later; for now observe that an `optimal(O)` query returns that (in this case, unique) instantiation of `O` which is highest in the `dom/2` order.

Example 2 (Holiday Planning, [Wil04]). There are three features: `time`, with values `l` and `n` for later and now; `place`, with values `m` and `o` for Manchester and Oxford, and `mode` with values `f` and `d` for fly and drive. The preference "All else being equal, I would prefer to go to Manchester" is formulated as clause 1 below. The rule "I would prefer to fly rather than drive, unless I go later in the year to Manchester, where the weather will be warmer, and a car would be useful for touring around" translates to clauses 2-3. The CP-theory rule "I would prefer to go next week, regardless of other choices." corresponds translated to clause 4, and the comparative preference rule "All other things being equal, I would prefer to fly now, rather than to drive later." to clause 5:

```
time(n). time(l). place(o). place(m). mode(f). mode(d).
outcome(o(T,P,M)):- time(T), place(P), mode(M).
/*1*/ d(o(X,m,Y),o(X,o,Y)).
/*2*/ d(o(T,P,f),o(T,P,d)):- T=n;P=o.
/*3*/ d(o(l,m,d),o(l,m,f)).
/*4*/ d(o(n,_,_),o(l,_,_)).
/*5*/ d(o(n,X,f),o(l,X,d)).
```

Proposition 1 (Normal form for Flat LCP rules). *Let R be a flat LCP-rule* `d(o(`$X_1,\dots,X_n$`), o(`$Y_1,\dots,Y_n$`)) :- c.` *where* `c` *is an equality constraint. For appropriate choices of disjoint index sets J, K, M and Z s.t. $\{1,\dots,n\} = J \cup K \cup M \cup Z$, and given L and U disjoint subsets of M and given constants $v_j (j \in J)$, $a_m, a'_m (m \in M, a_m \neq a'_m)$, $a_l (l \in L)$ and $a_u (u \in U)$, R is logically equivalent to the clause* `d(o(`$S_1,\dots,S_n$`), o(`$T_1,\dots,T_n$`)).` *where S_i, T_i are defined by: $S_j = T_j = v_j (j \in J)$, $S_k = T_k, k \in K$, $S_m = a_m, T_m = a'_m (m \in M)$, $S_z = X_z, T_z = Y_z (z \in Z)$, $S_l = X_l, T_l = a_l (l \in L)$, $S_u = a_u, T_u = Y_u (u \in U)$.*

The set J corresponds to the parent variables, the set K to the ceteris paribus variables, the set M to the variables that change the value from S to T and Z to the variables that are less important than the variables in M. We call L the lower-bound *set, and U the* upper-bound *set.*

Example 3. Given three variables `main`, `drink` and `dessert` with domains {`meat,fish,veg`}, {`water,wine`} and {`cake,fruit`} respectively. The rule "If I eat cake as dessert I prefer to drink water and I prefer to not eat meat", gives `Z,U,K=`$\emptyset$, `J=`{`dessert`}, `M=`{`main, drink`}, and `L=`{`A`} (clause 1). The rule "Given the same dessert, I always prefer fish as main course, regardless of the drink.", gives `J,L=`$\emptyset$ `Z=`{`drink`}, `K=`{`dessert`}, and `M=U=`{`main`} (clause 2):

```
/*1*/ d(o(X1, water, cake), o(meat, wine, cake)).
/*2*/ d(o(fish, X2, X3),o(Y1, Y2, X3)).
```

A GCP-net rule is simply a Flat LCP-rule such that $Z = \emptyset$, $|M| = 1$, $L \cup U = \emptyset$. A CP-theory rule is a Flat LCP-rule with $|M| = 1$, $L \cup U = \emptyset$. A comparative preference rule is a Flat LCP-rule with $L \cup U = \emptyset$.

The following theorems establish that LCP-theories conservatively extend these sub-languages. Proofs are straightforward, we have essentially just used standard logical notions to formalize the sub-languages:

Theorem 1 (Logical characterization of *ceteris paribus and general ceteris paribus*). *Given a CP-net $\mathcal{R}$, consider the set P of flat LCP-rules which represent all rows of its CP-tables. Then, for any two outcomes s and t, $s \succ t$ in $\mathcal{R}$ iff $\mathcal{L}(P) \vdash$* `dom(s,t)`*.*

Theorem 2 (Logical characterization of *CP-theories* and *comparative preference languages*). *Given a CP-theory $\mathcal{R}$, consider the set P of flat LCP rules modeling all the rules of the CP-theory. Given two outcomes s and t, $\mathcal{R} \vdash s \succ t$ iff $\mathcal{L}(P) \vdash$* `dom(s,t)`*.*

3.1 Recursive LCP-theories

Recursive or dependent rules are particularly useful in multi-agent contexts, where different agents may influence each other by stating their preferences depending on the preferences of some other agent [MMP$^+$13]. Here we illustrate with an extension to the Holiday planning example:

Example 4 (Holiday planning). John and Mary work for the same office and need to travel separately. "If John prefers Oxford to Manchester (all other things being equal), then Mary prefers Manchester to Oxford (all other things being equal)."

```
jPlace(X,Y) :- dom(o(J1, X, J3, M1, M2, M3), o(J1, Y, J3, M1, M2, M3)).
d(o(J1, J2, J3, M1, m, M3), o(J1, J2, J3, M1, o, M3)).
```

The predicate `jPlace(X,Y)` may be read as "John prefers place `X` to place `Y`, all other things being equal." The second clause may be read as saying "Mary prefers `m` to `o`, all other things being equal, provided that `jPlace(o,m)` holds.

Example 5. Let us consider two agents ranking features "appetizer" (rolls or bread), "main dish" (pasta or fish) and dessert (tiramisu or bread-pudding). We can formulate "If Alice doesn't prefer pasta, I would like to take pasta" as:

```
d(o(AA, AM, AD, MA, pasta, MD), o(AA, AM, AD, MA, fish, MD)) :-
   dom(o(_, fish,_,_,_,_),o(_, pasta,_,_,_,_)).
```

Note we do not assume acyclicity in variable ordering.

4 Algorithmic properties

The main algorithmic tasks regarding a preference theory are *dominance queries*, *consistency checking*, and *outcome optimization* (also of interest are *ordering queries*, but we rule them out of scope because of limitations of space). Below we fix a set of features `Var` with cardinality N and an LCP-theory P over `Var`.

Checking the dominance over a pair of outcomes corresponds to finding a swapping sequence in CP-theories or a flipping sequence in CP-nets. For LCP-theories this is determined by first-order derivability: the dominance query `(s,t)` succeeds iff $\mathcal{L}(P) \vdash$ `dom(s,t)`.

The problem of consistency checking is checking whether there is any outcome `s` such that $\mathcal{L}(P) \vdash$ `dom(s,s)`.

The problem of outcome optimization corresponds to find the most preferred outcome given an assignment to a (possibly empty) subset of features.

Definition 7 (Optimal outcome). *An* optimal outcome *is an outcome* `s` *such that there is no other outcome* `t` *such that* $\mathcal{L}(P) \vdash$ *`dom(s,t)` (i.e.* `s` *is an undominated outcome).*

Definition 8 (Optimal completion). *Given a (possibly non-ground) term* `s` *(representing a partial outcome) an* optimal completion *of* `s` *is a ground term* `t` *instantiating* `s` *s.t. for no other ground term* `t1` *instantiating* `s` *is it the case that* $\mathcal{L}(P) \vdash$ *`dom(t1,t)`.*

Note that an acyclic CP-net and CP-theory has a unique optimal outcome, but an LCP-theory may have several optimal outcomes.

Our algorithmic approach is based on analyzing whether the input LCP-theory corresponds to a special case (e.g. acyclic CP-net, tree-structured dependency graph, etc.). If so, optimal algorithms for the special case are used. Otherwise general Datalog procedures are used.

In the following sections we analyze the algorithms for dominance, consistency and optimality from a general point of view, and then considering the special cases in which the algorithms are faster. We take the viewpoint of *combined complexity*, i.e. assuming N, the clauses of the LPC theory, and the given query are supplied at runtime.

The results are summarized in Table 1.

	General structure	Acyclic	Tree
Dominance			
CP-nets:	PSPACE-comp [GLTW08]	PSPACE-comp [GLTW08]	Polynomial [BBD+04b,BFMZ13]
CP-theories:	PSPACE-comp [Wil04]	PSPACE-comp [Wil04]	**Polynomial**
Flat LCP-theories:	**PSPACE-comp**	**PSPACE-comp**	**?**
Recursive LCP-theories:	**EXPTIME-comp**	**PSPACE-comp**	**?**
Consistency/Optimality			
CP-nets:	PSPACE-complete [GLTW08]	Polynomial [BBD+04b]	Polynomial [BBD+04b]
CP-theories:	PSPACE-complete [Wil04]	Polynomial [Wil04]	Polynomial [Wil04]
Flat LCP-theories:	**PSPACE-complete/?**	**Polynomial***	**Polynomial ***
Recursive LCP-theories:	**EXPTIME-complete**	**PSPACE-complete**	**?**

*Bold results are provided in this paper, *with some constraint on the form of the rule, ? means the corresponding problem is open.*

Table 1. Computational complexity of Dominance, Consistency and Optimality

It is important to notice that for Recursive LCP-theories optimality and consistency procedures never have a lower computational complexity then the dominance procedure, because a recursive LCP rule also contains a dominance query.

We note in passing that for Flat LCP theories, data-complexity is also of interest. Recall that data-complexity for a Datalog programs is the complexity of determining, for a fixed program P, and input database D and query q, whether $P, D \vdash q$ (as a function of the size of D and q).

What is the distinction between P and D for LCP theories? For Flat LCP theories, P is simply the clauses for `dom/2`, and `consistent/0`. Once N, the number of features is fixed, this program is fixed. Thus data complexity for consistency of LCP theories corresponds to the complexity of determining for *fixed* N, whether $P, D \vdash$ `inconsistent`, as a function of the number of rules in the program. For Flat LCP (=linear Datalog) the data-complexity is NLogSpace-complete (see e.g. [GP03]).

4.1 Dominance

Theorem 3. *Given a flat LCP theory P over N features, deciding* `dom(s,t)` *is* PSPACE*-complete in N.*

Proof. Since flat LCP theories can encode the GCP-nets of [GLTW08], the dominance problem is at least PSPACE-hard. That the problem is in PSPACE can be established in a form similar to the proof of Theorem 4.4 in [GP03]. Since P has but a single rule, and the rule is linear, we can build up the proof non-deterministically using a polynomial amount of space. In fact, we need to keep space only for two ground facts of the form `dom(s, t)` where the `s,t` are constants. We start by using the base clause for `dom/2` to non-deterministically establish a `dom/2` fact, using some fact for `d/2`, and scratch space linear in N. Then we use the recursive clause for `dom/2` to non-deterministically generate a new `dom/2` fact. From this new fact, we can generate another, and delete the old fact. We stop when `dom(s,t)` is established.

Theorem 4. *Given a recursive LCP theory P over N features, deciding* `dom(s,t)` *is* EXPTIME*-complete in N.*

Proof. The proof is as above, except that the `d/2` clauses may no longer be linear, hence the combined complexity for full Datalog comes into play.

We note in passing – and document in the supplementary material of this paper – that the connection with Datalog allows for a simple and direct proof of the PSPACE-hardness of dominance for CP-nets. We show that the well-known PSPACE-HARD problem of determining whether a deterministic Turing Machine can accept the empty string without ever moving out of the first k tape cells can be reduced to checking dominance queries for CP-nets by modifying slightly the proof for Datalog in [GP03, Theorem 4.5].

In practice, the solutions of the dominance problem can be found using tabling on the `dom/2` predicate.

Note that in tree structured CP-nets a dominance query can be computed in time linear in N [BBD$^+$04b,BFMZ13]. We observe that a procedure similar to [BFMZ13] can be used for CP-theories. We use the generalized dependency graph G described in [Wil04] (see Section 2.2, this includes “importance” edges and dependency edges), and when G is a tree, we apply the dominance procedure of [BFMZ13].

4.2 Consistency and Outcome optimization

Consistency is determined by invoking theq query `?-consistent.` This takes advantage of the tabling of the `dom/2` predicate.

The following theorem affirms that consistency remains in PSPACE even when the language for preferences is extended beyond CP-nets to flat LCP rules, and it is a generalization of Theorem 3 in [GLTW08].

Theorem 5. *Given a flat LCP theory P over N features, deciding consistency is* PSPACE*-complete in N.*

The proof follows directly from the proof of Theorem 3 since consistency is reduced to checking entailment.

Theorem 6. *Given a recursive LCP theory P over N features, deciding consistency is* EXPTIME*-complete in N.*

As above, noting that the combined complexity for full Datalog is EXPTIME.

For optimality, the user invokes the query `?- optimal(s).` Note that `optimal/1` uses partial order answer subsumption (see Section 2.3). In theory this may result in an exponential number of calls to `dom/2` atoms, with each check taking exponential time. This leads to:

Theorem 7. *Given a recursive LCP theory P over N features, deciding optimality is in* EXPTIME *over N.*

For now we leave as open the corresponding problem for flat recursive theories (note that this problem is PSPACE-complete for CP-nets).

Optimization and Consistency for acyclic dependency graphs In acyclic CP-nets the sweep-forward procedure [BBD$^+$04b] finds the unique optimal outcome (or completion) in polynomial time. A similar result holds for [Wil04].

Under the assumptions of consistency and acyclicity, an optimal outcome (and completion) can also be found for Flat LCP theories in polynomial time, using the following algorithm *Acyclic-LCP-Opt* generalizing sweep-forward (see Section 5 for implementation details):

1. Given a set of LCP-rules we compute the dependency graph G, and we check if G is acyclic.
2. We compute a total order $\mathcal{O} = \{X_1, \ldots, X_N\}$ over `Var` as a linearization of the topological order defined by G.
3. For each variable X_i, chosen following $\mathcal{O}$, we consider a set $W_i \subseteq$ `Var` such that it contains all the variables that change the value jointly with X_i in at least one rule (W_i could contain only X_i). Using the rules in the LCP-theory that involve the variables in W_i, and given the assignments for the variables $X_1, \ldots, X_{i-1}$, we generate an ordering over the partial outcomes defined on the variables in W_i. We assign to X_i the X_i value of the top element of the ordering that satisfies `outcome/1` for at least one completion (the completion has the given assignment to $X_1, \ldots, X_{i-1}$ and an arbitrary assignment to $\{X_{i+1}, \ldots, X_N\} \setminus W_i$).
4. We repeat the previous step for all the features in `Var` (following $\mathcal{O}$), never changing the value of an assigned variable.

Proposition 2. *The outcome obtained with the* Acyclic-LCP-Opt *procedure is an optimal outcome for acyclic LCP-theories.*

Proof. We suppose by contradiction that exists an outcome o' such that $o' \succ o$. This implies that exists a chain of outcomes $o' = o_1, \cdots, o_m = o$ such that $\forall i \in \{1, \cdots, m-1\}$ exists a rule R that implies $o_i \succ o_{i+1}$. Considering a linearization $X_1, \cdots, X_N$ of the graph G associated to our set of rules $\mathcal{C}$, we take the first variable X in this order such that $o' \restriction_X \neq o \restriction_X$. Thus there exist a rule R and an index $i \in \{1, \cdots, m-1\}$ such that $o' \restriction_X = o_1 \restriction_X = \cdots = o_i \restriction_X \neq o_{i+1} \restriction_X = \cdots = o_m \restriction_X = o \restriction_X$ and $o_i \succ o_{i+1}$. This implies that $o \restriction_X$ is not the maximal element in the set of rules that involve X, that is a contradiction.

In LCP-theories it is possible to have many different optimal outcomes: we can obtain the whole set of optimal outcomes using the *Acyclic-LCP-Opt* procedure. If there is more then one optimal outcome this means that there exists some variable X that in the third step of the algorithm has more then one top element. Running in parallel all these possible assignments we obtain the whole set of optimal outcomes.

The complexity of the procedure is $O(d^w * N)$ where $w = \max_i |W_i|$: the third step of the procedure could involve all the partial outcomes defined on W_i. If the program bounds w, the procedure become linear in N. Note that if the LCP-theory corresponds to a CP-net or a CP-theory then $|W_i| = 1 \ \forall i$ and the algorithm coincides with the sweep-forward procedure for CP-nets, and the procedure introduced in [Wil04] for CP-theories.

We can use this procedure also to compute the optimal completion of a given partial outcome, simply considering the partial outcome as pre-assigned values for a subset of features.

Recursive LCP-theories may require m dominance queries, where m is the number of dominance goals in the body of the input preference rules. Since we use tabling, the time cost is amortized over all calls from the problem-solver (in lieu of space). With this change, the procedure described above can be used. Because of the dominance queries, the optimality procedure is PSpace-complete.

4.3 Decreasing complexity using evidence specification

We note that if evidence for some variables is given, the resulting simplified LCP theory may have the structure of one of the special cases discussed above, and hence optimality and dominance queries may be answered using specialized, polynomial procedures. To this end, the implementation needs to maintain a dynamic dependency graph that ignores features for which values have been provided.

5 Experimental evaluation

We have developed a compiler for LCP theories, also called LCP. The compiler and associated tooling will be made avaiable on Github as an open source project under the Eclipse Public Licence.

The compiler reads an LCP-theory, builds the dependency graph and checks whether it represents an acyclic CP-net. If so, it performs a linearization of the dependency graph, and produces a pre-digested representation of the theory. Otherwise it emits the clauses unchanged so that the standard default (tabled) algorithms optimality, consistency and dominance can be used.

In more detail, the compiler captures (a linearization of) the dependency order in a clause

`dependency(`$[a_1, a_2, \ldots, a_N]$`).`

where $a_i \in 1 \ldots N$ (features are implemented as Prolog integers), and if a_j depends on a_i, then $i < j$. Suppose a_p depends on $a_{i_1}, \ldots, a_{i_k}$. If the input LCP program specifies that if each of the features $a_{i_1}, \ldots, a_{i_k}$ had values $x_{i_1}, \ldots, x_{i_k}$ respectively, then the known order of values of a_p is given by (best) $w_1, \ldots, w_r$ (worst), then the compiler emits the fact

`preference(`$[x_{p-1}, \ldots, x_1]$`, `$[w_1, \ldots, w_r]$`).`

with $x_{i_1}, \ldots, x_{i_k}$ as constant, and the remaining x_i as unique variables (occur only once in the clause). Thus we use Prolog unification to select the correct preference clause to use, given the current partial outcome `[v_p, ..., v_1]` specifying values for the first `p` attributes (in reverse dependency order).

The following code for `optimize/1` uses these clauses and implements *Acyclic-LCP-Opt*:

```
optimize(O) :-
   O=o(_,_), dependency(D), reorder(D, O, AList), optimize_a([], AList).
reorder([], _, []).
reorder([X| Xs], O, [V | Vs]) :- arg(X, O, V), reorder(Xs, O, Vs).
optimize_a(_, []).
optimize_a(Upargs, [Xs|R]) :-
   select(Upargs, Xs), append(Xs, Upargs, Upargs1), optimize_a(Upargs1, R).
select(Us, []).
select(Us, [X | R]) :- nonvar(X), select(Us, R).
select(Us, [X | R]) :- var(X), preference(Upargs, [X|_]), select(Us ,R).
```

We have also implemented a CP-net generator. Generating CP-nets i.i.d. is non-trivial [AGM14] and therefore we use an approximation method that randomly generates acyclic CP-nets with N features, given a maximum number of dependencies for each feature. We consider a fixed ordering $X_1, \ldots, X_N$ of features. We also take as input the maximum in-degree for each feature, k. We first generate the acyclic dependency graph. For each feature X_i, we randomly choose its in-degree $d \in 0..\min\{k, i-1\}$. Next, we randomly choose d parents from the features $\{X_1, \ldots, X_{i-1}\}$. When the graph is built, we fill in the CP-tables choosing randomly one element of the domain (since the domain is binary). The resulting CP-net is written out as an LCP theory, using XSB Version 3.5.0 syntax [SW12b].

We have run two different kinds of experiment: in the first with a fixed upper bound for the number of dependencies for each feature, we varied the number of features from 5 to 200 and measure running time for optimality queries. In the second experiment, fixing the number of features, we varied the upper bound of dependencies from 1 to 10. In both experiments we asked for the optimal outcome 100 times and then we computed the average elapsed time to output the result.

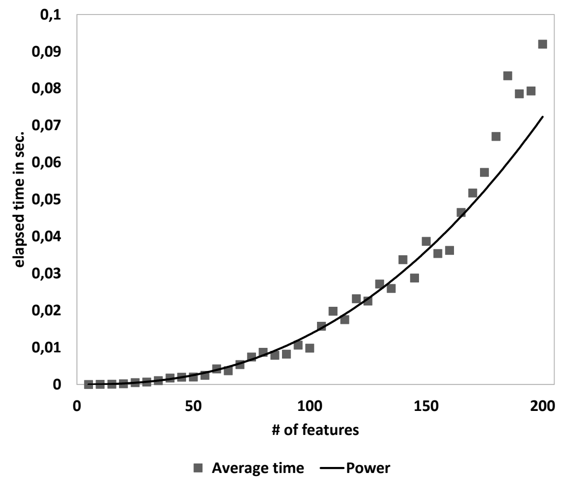

Fig. 1. Optimal outcome performances.

Figure 1 shows the results for the experiment where the upper bound for the number of dependencies is fixed to 6. The elapsed time to compute the optimal outcome grows quadratically in the number of features. This is in line with our results as summarized in Table 1. (The runtime is not linear because each step involves checking for the value of parents using unification on $O(N)$ terms.)

6 Conclusion and future work

We have presented a new framework for conditional preferences, based on expressing preferences using Datalog. We have shown how dominance, consistency and optimality queries can be formulated directly in Datalog and implemented in modern tabled Prolog systems such as XSB Prolog. We have also analyzed the complexity of the different algorithms, developed efficient procedures for some common use cases, and implemented a translator that exploits these algorithms.

Much work lies ahead. Table 1 contains some open complexity questions. We hope to exploit Datalog theory to develop more efficient special cases. We hope to use the implemented LCP system in real-life applications to determine the adequacy of the LCP system.

Acknowledgements. We gratefully acknowledge conversations with David S. Warren and Terrance Swift. These conversations led to a small code-change in the XSB compiler to better support LCP theories. We also acknowledge conversations with Francesca Rossi.

References

[AGM14] T.E. Allen, J. Goldsmith, and N. Mattei. Counting, ranking, and randomly generating CP-nets. In *In Proceedings of the 8th Multidisciplinary Workshop on Advances in Preference Handling (MPREF)*, 2014.

[BBD$^+$04a] C. Boutilier, I. Brafman, C. Domshlak, H. Hoos, and D Poole. Preference-based Constrained Optimization with CP-nets. *Computational Intelligence*, 20(2):137–157, 2004.

[BBD$^+$04b] C. Boutilier, R.I. Brafman, C. Domshlak, H.H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[BFMZ13] D. Bigot, H. Fargier, J. Mengin, and B. Zanuttini. Probabilistic conditional preference networks. In *Proc. of the 29th International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

[CGT89] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Trans. on Knowl. and Data Eng.*, 1(1):146–166, March 1989.

[DB02] C. Domshlak and R.I. Brafman. CP-nets: Reasoning and consistency testing. In *Proc. 8th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2002.

[DEGV01] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, September 2001.

[DPR$^+$06] C. Domshlak, S. Prestwich, F. Rossi, K. Venable, and T. Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *J. Heuristics*, 12(4-5):263–285, 2006.

[DRVW03] C. Domshlak, F. Rossi, K.B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[FV99] T. Feder and M. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study Through Datalog and Group Theory. *SIAM J. Comput.*, 28(1):57–104, February 1999.

[GLTW08] J. Goldsmith, J. Lang, M. Truszczynski, and N. Wilson. The Computational Complexity of Dominance and Consistency in CP-nets. *Journal of Artificial Intelligence Research*, 33(1):403–432, 2008.

[GP03] G. Gottlob and C. Papadimitriou. On the complexity of single-rule datalog queries. *Inf. Comput.*, 183(1):104–122, May 2003.

[KKR95] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26 – 52, 1995.

[MMP$^+$13] A. Maran, N. Maudet, M. S. Pini, F. Rossi, and K. B. Venable. A Framework for Aggregating Influenced CP-nets and Its Resistance to Bribery. In *Proceedings of AAAI-27*, pages 668–674, 2013.

[PRVW04] S. Prestwich, F. Rossi, K. B. Venable, and T. Walsh. Constrained CP-nets. In *in Proceedings of CSCLP04*, 2004.

[RVW04] F. Rossi, K.B. Venable, and T. Walsh. mCP nets: representing and reasoning with preferences of multiple agents. In *Proc. of the 19th AAAI Conference on Artificial Intelligence (AAAI)*, 2004.

[SW10] T. Swift and D. S. Warren. Tabling with answer subsumption: Implementation, applications and performance. In *Proceedings of the 12th European Conference on Logics in Artificial Intelligence*, JELIA'10, pages 300–312, Berlin, Heidelberg, 2010. Springer-Verlag.

[SW12a] T. Swift and D. S. Warren. XSB: Extending Prolog with Tabled Logic Programming. *Theory Pract. Log. Program.*, 12(1-2):157–187, January 2012.

[SW12b] T. Swift and D. S. Warren. XSB home page. `http://http://xsb.sourceforge.net/`, 2012.

[Tom98] D. Toman. Memoing Evaluation for Constraint Extensions of Datalog. *Constraints*, 2(3/4):337–359, January 1998.

[Var82] M. Vardi. The complexity of relational query languages (extended abstract. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC 82*, pages 137–146, 1982.

[Wil04] N. Wilson. Extending CP-Nets with Stronger Conditional Preference Statements. In *Proceedings of AAAI-04*, pages 735–741, 2004.

[Wil09] N. Wilson. Efficient Inference for Expressive Comparative Preference Languages. In *Proceedings of IJCAI-09*, 2009.